KaSim reference manual v1.05

J. Krivine

November 3, 2010

This document is work in progress...

Contents

Chapter 1

Introduction

1.1 The KaSim engine

KaSim is an open source stochastic simulator of rule-based models [3, 2, 4] written in the κ-calculus. Basically KaSim takes one or several kappa files as inputs and generates stochastic trajectories of various observable. KaSim implements the network free simulation algorithm for rule-based models [1] that extends Gillespie's algorithm [5, 6].

A *simulation event* corresponds to the application of a rewriting rule, contained in the kappa file, to the current graph (also called a *mixture*). The rule is selected according to its *activity*, *i.e* the number of instances it has in the current mixture, multiplied by its kinetic rate, and applied one of its possible instance in the graph. It results in a new graph together with an updated activity for all rules (see Fig. 1.1).

Importantly, the cost of such an event is independent of the size of the graph it is applied to [1]. Note that KaSim is not equipped with a curve visualization tool. However, the outputted data are in text format and usable with any standard plotting software such as GnuPlot.

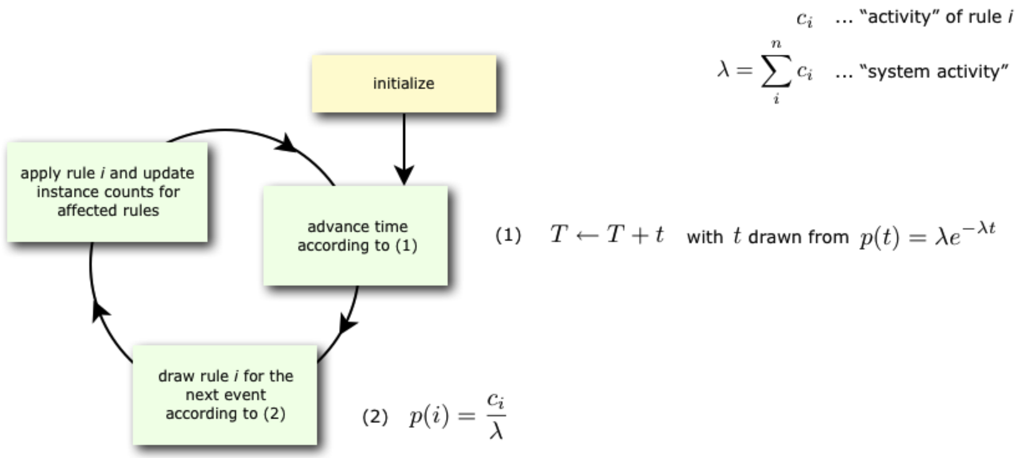

Figure 1.1: The event loop

1.2 Support

- Kappa language tutorials and downloads: `http://kappalanguage.org`
- Bug reports should be posted on github: `https://github.com/jkrivine/KaSim/issues`
- Questions and answers on the kappa-user mailing list: `http://groups.google.com/group/kappa-users`
- Want to contribute to the project? `jkrivine` at `pps` dot `jussieu` dot `fr`

Chapter 2

Installation

2.1 Obtaining the sources

To obtain KaSim you can either use pre-compiled binaries available on KappaLanguage.org or compile the sources for your architecture. To do so, download the source code from `https://github.com/jkrivine/KaSim` and make sure you have a recent ocaml compiler installed. From a terminal window type `ocamlopt.opt -v`. If nothing appears then you need to install Ocaml Native compiler that can be downloaded from `http://caml.inria.fr/download.en.html`.

2.2 Compilation

Once Ocaml is safely installed, untar KaSim archive and compile following these few steps:

```
$ tar xzvf kasim.tar.gz -d Kappa

$ cd Kappa

$ make
```

At the end of these steps you should see, in the `Kappa` directory, an executable file name KaSim. In order to check the compilation went fine, simply type `.\KaSim --version`. If the ocaml native compiler `ocamlopt.opt` is not in the path of your system, you may set the variable `OCAMLBINPATH` to point to the location of the compiler by editing the corresponding line in the Makefile.

Chapter 3

The command line

3.1 General usage

From a terminal window, KaSim can be invoked by typing

```
$ KaSim -i file_1 ...  -i file_n [option]
```

where `file_i` are the input kappa files containing the rules, initial conditions and observable see Chapter 4 below. Tables 3.1 and 3.2 summarize all the options that can be given to the simulator. Basically one should specify an upper bound either in bio time (arbitrary time unit), or number of events. Note that bio-time is computed using Gillespie's formula for time advance (see Fig. 1.1) and should not be confused with CPU-time (it's not even proportional). In doubt we recommend using a bound in number of events since the cost of one event application is bounded (in CPU time) by a constant, so the simulation time of n events is roughly k times the simulation time of $k * n$ events.

3.2 Main options

Table 3.1: Command line: main options

Argument	Description
`-e` e_{max}	Terminates simulation after $e_{max} \geq 0$ events
`-t` t_{max}	Terminates simulation after $t_{max} \geq 0.0$ time units
`-p` n	Produces a data file (default: `data.out`) with $n \geq 0$ data points
`-o` *file*	Set the name of data file to *file*
`-d` *dir*	Redirects all output files to the directory *dir*

3.3 Advanced options

Table 3.2: Command line: advanced options

Argument	Description
`--seed` n	Seeds the pseudo-random number generator $n > 0$
`--implicit-signature`	Automatically deduce agent signatures

Example

The command

```
$ KaSim -i model.ka -e 1000000 -p 1000 -o model.out
```

will generate a file `model.out` containing the trajectories of the observables defined in the kappa file `model.ka`. The file `model.out` will contain a 1000 data point (*i.e* in this case, a measure will be taken every 1000 events). The command

```
$ KaSim -i init.ka -i rules.ka -i obs.ka -i mod.ka -t 1.5 -p 1000
```

will generate a file `data.out` (default name) containing 1000 data points of a simulation of 1.5 seconds (arbitrary time units) of the model. Note that the input kappa file is splited in 4 files containing, for instance, the initial conditions, `init.ka`, the rule set, `rules.ka`, the observables, `obs.ka`, and the perturbations, `pert.ka` (refer to Chapter 4 for details). Note that the order in which the files are given does not matter.

Chapter 4

The kappa file

4.1 General remarks

The *Kappa File* (KF) is the formal representation of your model. We use KF to denote the union of the files that are given as input to KaSim (argument `-i`). Each line of the KF is interpreted by KaSim as a *declaration*. If the line is ended by the escape character '\' the continuation of the declaration is parsed onto the next line. Declarations can be of 4 types: *signatures* (Sec. 4.2), *rules* (Sec. 4.3), *variables* (Sec. 4.4), *initial conditions* (Sec. 4.5) and *perturbations* (Sec. 4.6). The KF's structure is quite flexible since it can be divided in any sub-files and the order in which declarations are entered does not matter.

4.2 Agent signature

Agent signatures constitute a form of typing information about the agents that are used in the model. It contains information about the name and number of interaction sites the agent has, and about their possible internal states. A signature is declared in the KF by the following line:

`%agent:` *signature_expression*

according to the grammar given Table 4.1. For instance the line:

signature_expression	::=	**Id** **(***interface_decl***)**
interface_decl	::=	**Id** *internal_state_decl***,** *interface_decl* | ε
internal_state_decl	::=	~**Id** *internal_state_decl* | ε

Table 4.1: Agent signature expression: terminal symbol **Id** can be any string generated by regular expression [*a-z* *A-Z* 0-9][*a-z* *A-Z* 0-9 $-$ _]*

```
%agent:  A(x,y~u~p,z~0~1~2)
```

will declare an agent `A` with 3 *(interaction) sites* `x,y` and `z` with the site `y` possessing two *internal states* `u` and `p` (for instance for the unphosphorylated and phosphorylated forms of `y`) and the site `z` having possibly 3 states respectively 0, 1 and 2. Note that internal states values are purely conventional.

4.3 Rules

4.4 Variables

4.5 Initial conditions

4.6 Perturbation language

Bibliography

[1] Vincent Danos, Jérome Féret, Walter Fontana, and Jean Krivine. Scalable simulation of cellular signaling networks. In *Proceedings of APLAS'07: $5^t h$ ASIAN symposium on programming languages and systems*, volume 4807 of *LNCS*, pages 139–157, 2007. Invited paper.

[2] Vincent Danos, Jérôme Feret, Walter Fontanta, Russ Harmer, and Jean Krivine. Rule based modeling of biological signaling. In Luís Caires and Vasco Thudichum Vasconcelos, editors, *Proceedings of CONCUR 2007*, volume 4703 of *LNCS*, pages 17–41. Springer, 2007.

[3] Vincent Danos and Cosimo Laneve. Formal molecular biology. *Theoretical Computer Science*, 325, 2004.

[4] James R. Faeder, Mickael L. Blinov, and William S. Hlavacek. Rule based modeling of biochemical networks. *Complexity*, pages 22–41, 2005.

[5] Daniel T. Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4):403–434, 1976.

[6] Daniel T. Gillespie. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, 81(25):2340–2361, 1977.

Index